Monte Carlo Methods and Appl., Vol. 12, No. 5-6, pp. 385 – 393 (2006)

A Repetition Test for Pseudo-Random Number Generators

Manuel Gil, Gaston H. Gonnet, Wesley P. Petersen

SAM, Mathematik, ETHZ, HG G-52.1
Zurich, CH8092

Abstract — A new statistical test for uniform pseudo-random number generators (PRNGs) is presented. The idea is that a sequence of pseudo-random numbers should have numbers reappear with a certain probability. The expectation time that a repetition occurs provides the metric for the test. For linear congruential generators (LCGs) failure can be shown theoretically. Empirical test results for a number of commonly used PRNGs are reported, showing that some PRNGs considered to have good statistical properties fail. A sample implementation of the test is provided over the Internet.

Introduction

Empirical statistical tests are an important tool when the decision has to be made whether a sequence generated by a PRNG is sufficiently random and useful for a certain application. A large number of tests like the ones described in Knuth's book [4], Marsaglia's Diehard battery [10], L'Ecuyer's TestU01 suite [8] or the NIST battery [12] are available. Even if a generator behaves randomly with respect to a number of tests, one can not be sure that it will not fail in a further test. Each test therefore increases the confidence in the randomness of the produced sequence. The *Repetition Test* described here tests a PRNG to see if the produced numbers reappear at the expected time distribution. It has some similarity to the birthday problem. It should be noticed that a repetition does not necessarily coincide with the period of the sequence (unless the PRNG is a one-step recurrence like for example an LCG).

Repetition Time: Expected Value and Variance

In this section the mathematical background of the Repetition Test is introduced in terms of a ball and urn experiment. Let an urn contain n differently numbered balls and consider a sampling with replacement experiment, where each ball is equally likely to appear. Balls are sampled from and returned to the urn until a ball is picked up that already appeared. This event will be called *first repetition*, and the trial r at which it happens is the *repetition time*.

The probability P_i that no repetitions happen when the first i balls are picked up is

$$P_i = \sum_{j=i+1}^{n} p_j = \frac{n(n-1)(n-2)\cdots(n-i+1)}{n^i} = \frac{n!}{n^i(n-i)!} \ , \tag{1}$$

where p_i is the probability of a repetition at time i. The expected repetition time $E[r]$ can be expressed as a summation of the P_i

$$E[r] = \sum_{i=0}^{n} i p_i = \sum_{i=0}^{n} \sum_{j=i+1}^{n} p_j = \sum_{i=0}^{n} P_i \ .$$

This leads to the following expression

$$E[r] = \sum_{i=0}^{n} \frac{n!}{n^i(n-i)!} \ . \tag{2}$$

The second moment of r is

$$E[r^2] = \sum_{i=0}^{n} i^2 p_i = \sum_{i=0}^{n} \left((i+1)^2 - i^2\right) \sum_{j=i+1}^{n} p_j = \sum_{i=0}^{n} (2i+1) P_i \ ,$$

which simplifies to

$$E[r^2] = 2n + E[r] \ ,$$

hence the variance is given by

$$\mathrm{Var}(r) = 2n + E[r] - E[r]^2 \ . \tag{3}$$

Knuth [4] (section 1.2.11.3) provides an asymptotic expression for the function $Q(n) = E[r] - 1$ and we get

$$E[r] = \sqrt{\frac{\pi n}{2}} + \frac{2}{3} + \frac{1}{12}\sqrt{\frac{\pi}{2n}} - \frac{4}{135n} + \frac{1}{288}\sqrt{\frac{\pi}{2n^3}} + O(n^{-2}) \ . \tag{4}$$

$$\mathrm{Var}(r) = \left(2 - \frac{\pi}{2}\right) n + \sqrt{\frac{\pi n}{2}} + O(1) \tag{5}$$

Since in the Repetition Test n is usually very large the asymptotic approximations are very accurate and useful.

The quantities derived above are related to the birthday paradox. Assuming a uniform distribution of birthdays over a year of 365 days, the birthday paradox states that in a room of 23 randomly chosen people the probability p that at least two of them share a birthday is greater than 50%. We can compute this probability by noting that it is complementary to the one given in (1) for $n = 365$ and $i = 23$, i.e.

$$p = 1 - P_{23} = \frac{365!}{365^{23}(365-32)!} \approx 0.507 \; .$$

In the context of the birthday problem (2) denotes the expected number of people that have to come into a room so that a repeated birthday first appears, i.e.

$$E[r] = \sum_{i=0}^{365} \frac{365!}{365^i (365-i)!} \approx 24.62 \; ,$$

$$V[r] = 2 \cdot 365 + E[r] - E[r]^2 \approx 148.64 \; .$$

Determining the Repetition Time of PRNGs

This section describes the Repetition Test for PRNGs. It is an adaptation of the urn experiment described in the previous section.

For w-bit integer PRNGs that produce numbers between 0 and $2^w - 1$ the adaptation is straightforward. Under the hypothesis that the generated numbers are uniformly distributed and, according to (4), the first repetition should happen after the generation of $O(\sqrt{n}) = O(2^{w/2})$ integers. For floating point generators on the other hand some care has to be taken as the floating point numbers are not uniformly dense on the real line. An equal number of them fall between successive negative powers of 2. Uniformity can be assumed by sieving the generated numbers for a range with fixed mantissa $\left[2^k, 2^{k+1}\right)$. In the following this range is denoted by $[L, 2L)$. The number of possible values [1] for a w-bit mantissa is then $n = 2^w$. For example, a PRNG producing $U(0,1)$ distributed IEEE doubles can be sieved for the interval $[0.5, 1)$, in which case n is set to 2^{53}. Below an

[1] It can also be calculated by $n = \frac{1}{\epsilon_M}$, where ϵ_M is the smallest floating point value such that $1 + \epsilon_M > 1$.

algorithm to obtain empirically an estimate $\hat{r}$ of the expected repetition time of an PRNG is given. It determines N times the first repetition on disjoint subsequences and compares $\hat{r}$ to the expectation (4) for some confidence level c. Since $O(2^{w/2})$ numbers have to be stored in main memory to detect a repetition a hash table H of size M is used. M is set to the expected repetition time plus ten standard deviations. An overflow of the hash table then implies failure of the PRNG in the Repetition Test. Note that in an implementation of this algorithm one has to make sure that the generated numbers s are stored in main memory and not kept in registers with extra bits as this would alter equality comparisons.

```
"Seed the PRNG"
r̂ ← 0;  M ← E[r] + 10√Var[r]
for k ← 1 to N do
    for i ← 1 to M do H[i] ← −1 end
    t ← 0;  repetition ← false
    while not repetition do
        repeat s ← "Get random number from PRNG" until s ∈ [L, 2L)
        t ← t + 1
        if t > M then "Test failed"; abort
        hashFound ← false;  i ← (s/L mod M) + 1
        while not repetition and not hashFound do
            if H[i] = s then repetition ← true
            if H[i] = −1 then H[i] ← s;  hashFound ← true
            i ← i + 1
        end
    end
    r̂ ← r̂ + t
end
r̂ ← r̂/N
if |r̂ − E[r]| ≤ c√(Var[r]/n) then "Test passed" else "Test failed"
```

The following two propositions concern the repetition times of LCGs and MRGs.

Proposition 1. *As LCGs are one-step generators, their repetition time always coincides with their period, usually $O(2^w)$, so they fail the Repetition Test.*

Proof. An LCG generates numbers according to the recursion formula

$$s_n = as_{n-1} + b \ \mathbf{mod}\ m.$$

where s_n, a, b and m are integers. Obviously it can generate no more than m different numbers. As soon as a number s_i is repeated for the first time, there is a $p > 0$ such that $s_i = s_{i-p}$. The same period (of length p) which has been generated is started again such that for all $j > 0$ $s_{i+j} = s_{i-p+j}$. Therefore the repetition time coincides with p. The period depends on the choice of s_0, a, b and m. Usually these parameters are chosen such that the maximal period length is obtained. For example if the multiplier a is a primitive root modulo m (for prime m) then $p = m - 1$. For certain odd multipliers for $m = 2^w$ the period is 2^{w-1}. These and further results can be found in [4]. In any case an LCG is not going to pass the Repetition Test unless it was designed to have a short period of $O(2^{w/2})$ which is in itself an undesirable property. □

Lemma 1. *The repetition times of a multiple recursive generator (MRG) of the form $x_n = \sum_{i=1}^{l} a_i x_{n-i}$ that is initialized with a pure multiplicative LCG $s_n = bs_{n-1}$ and where all the arithmetic is performed modulo m are equal for all seeds.*

Proof. Initializing the MRG with an LCG means that $x_i = s_i$ for $1 \leq i \leq l$, where s_0 is the seed. Since $s_n = b^n s_0$, every number x generated by the MRG can be written as $x = s_0 f(b, a_1, a_2, \ldots, a_l)$, where f is a linear combination of the a_i and of powers of b. Consequently, whether two numbers in the pseudo-random sequence are equal or not does not depend on s_0. □

Note that the repetition time of PRNGs having a state vector of dimension greater than one (e.g. MRGs) does not coincide with the PRNG's period length. In that case a repetition of a number in the sequence does not imply a repetition of the entire sequence. In practical applications only a small part of the period of a PRNG is used. These samples should have numbers re-appear in sequences of length $O(2^{w/2})$.

Empirical Tests

The mean repetition time has been determined empirically for a number of commonly used PRNGs using an implementation of the algorithm described in the previous section. The program is coded in the C language and available from [2]. The variance (5) of the first repetition is quite large, so small samples are not very revealing. A sample size of $N = 100$ has been chosen and the mean repetition time has been evaluated for a 95% confidence level.

What follows is a very short description of the generators:

- **ran3** is a subtractive lagged-fibonacci generator from the Numerical Recipes [13].
- **rcarry** is a subtract-with-borrow generator [3].
- **ranlux** improves rcarry by throwing away a fraction of the generated numbers [9]. Only the highest luxury level version results are reported here.
- **ziff98** denotes a four-tap generalized feedback shift-register (GFSR) [14].
- **mt19937** is the Mersenne Twister, a 32-bit twisted GFSR from [11]. A `double` generator, obtained by dividing the integers by $2^{32}-1$, has also been tested. Another `double` variant with 53-bit resolution, obtained by shifting and concatenation of two 32-bit integers from mt19937, is denoted by mt19937_{53}.
- **ecuyer93** denotes the sample C-implementation of a fifth-order MRG presented in [5].
- **ecuyer96** denotes the sample C-implementation of a combined MRG presented in[6].
- **taus88** denotes the sample C-implementation of a tausworthe PRNG presented in [7].
- **ghg** [1] is a `double` PRNG of the form

$$x_n = 32 \cdot x_{n-19} + \frac{1}{128} \cdot x_{n-67} + x_{n-128} \ \ \textbf{mod}\ 1.0.$$

The computation is done entirely in floating point arithmetic.

Table 1 summarizes the results. The values are averages of 100 runs. The `float` and `double` generators have been tested for the $[0.5, 1)$ interval. There are 2^{32} 32-bit `integer`s and for a fixed mantissa 2^{23} `float`s or 2^{53} `double`s, so using (4) the expected repetition times for `float` [f], `double` [d] and `integer` [i] generators are $3.6307 \cdot 10^3$, $8.4108 \cdot 10^7$ and $8.2138 \cdot 10^4$. Failures with a 95% confidence are underlined.

Table 1: Results of repetition tests for three seeds.

seed:		331	717	1236
ran3	[f]	$3.56 \cdot 10^3$	$3.72 \cdot 10^3$	$3.52 \cdot 10^3$
rcarry	[f]	$3.40 \cdot 10^3$	$3.60 \cdot 10^3$	$3.53 \cdot 10^3$
ranlux	[f]	$3.41 \cdot 10^3$	$3.91 \cdot 10^3$	$3.74 \cdot 10^3$
mt19937	[d]	$\underline{5.86 \cdot 10^4}$	$\underline{5.73 \cdot 10^4}$	$\underline{6.06 \cdot 10^4}$
mt19937	[i]	$8.68 \cdot 10^4$	$8.58 \cdot 10^4$	$8.67 \cdot 10^4$
mt19937_{53}	[d]	$9.02 \cdot 10^7$	$8.66 \cdot 10^7$	$8.69 \cdot 10^7$
ziff98	[i]	$\underline{6.12 \cdot 10^4}$	$\underline{6.26 \cdot 10^4}$	$\underline{6.07 \cdot 10^4}$
ecuyer93	[i]	$\underline{5.91 \cdot 10^4}$	$\underline{5.58 \cdot 10^4}$	$\underline{5.65 \cdot 10^4}$
ecuyer96	[i]	$\underline{5.75 \cdot 10^4}$	$\underline{5.30 \cdot 10^4}$	$\underline{5.78 \cdot 10^4}$
taus88	[i]	$8.00 \cdot 10^4$	$8.42 \cdot 10^4$	$8.34 \cdot 10^4$
ghg	[d]	$8.46 \cdot 10^7$	$8.79 \cdot 10^7$	$8.96 \cdot 10^7$

All the failing generators repeat too early. Note that mt19937 as an integer generator passes the test, while as a double generator the repetitions happen too early. This is not much of a surprise as mt19937 has been designed to be a 32-bit PRNG. The mt19937_{53} variant which possesses a 53-bit resolution passes. One should be aware that good properties of a PRNG can be destroyed by simple transformations. Care has to be taken whenever an `integer` generator is converted to a `float` or `double` generator as the expected repetition time is different for different number systems. Furthermore there are several ways to perform the mapping and for a particular mapping the repetition properties are not necessarily propagated. In this case conclusions drawn from the `integer` sequence can not be applied to the `float` or `double` sequence.

Conclusion

An empirical statistical test called the Repetition Test has been proposed. It is based on a ball and urn model. One may judge whether repetitions happen to soon or to late by knowing the expected repetition time as well as its variance. Exact forms as well as an asymptotic expression were given. They are a function of the number of elements of the generators number system. Repetitions which always appear too soon are bad, but repetitions which are consistently too large are also undesirable. The latter is the case for LCGs, because they repeat only at their period. Sophisticated generators, for example combined MRGs, failed in empirical test by repeating too soon.

References

[1] Gonnet, G., Gil, M, Petersen, W.P.: Multiple Recursive Generators & The Repetition Test. Technical Report #476, Department of Computer Science, ETH Zürich (2005)

[2] Gonnet, G.: Repeating Time Test for U(0,1) Random Number Generators. `http://www.inf.ethz.ch/~gonnet/RepetitionTest.html`, ETH Zürich (2003)

[3] James, C.F.: A review of pseudorandom number generators. Computer Physics Communications, **60**, 329–344 (1990)

[4] Knuth, D.: The Art of Computer Programming, Addison Wesley, New York (1968)

[5] L'Ecuyer, P., Blouin, F., Coutre, R.: A search for good multiple recursive random number generators. ACM Transactions on Modeling and Computer Simulation, **3**, 87–98 (1993)

[6] L'Ecuyer, P.: Combined multiple recursive random number generators. Operations Research, **44**(5), 816–822 (1996)

[7] L'Ecuyer, P.: Maximally Equidistributed Combined Tausworthe Generators. Mathematics of Computation, **65**(213), 203–213 (1996)

[8] L'Ecuyer, P., Simard, R.: TestU01: A Software Library in ANSI C for Empirical Testing of Random Number Generators. `http://www.iro.umontreal.ca/~simardr/indexe.html` (2005)

[9] Lüscher, M.: A portable high-quality random number generator for lattice field theory simulations. Computer Physics Communications, **79**, 100–110 (1994)

[10] Marsaglia, G.: The Diehard Battery of Tests of Randomness. `http://www.stat.fsu.edu/pub/diehard`, Florida State University (1985)

[11] Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul., **8**(1), 3–30 (1998)

[12] National Institute of Standards and Technology: A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications. `http://csrc.nist.gov/rng` (2001)

[13] Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: Numerical Recipes: The Art of Scientific Computing. Cambridge University Press, Cambridge (UK) and New York (1992)

[14] Ziff, R.M.: Four-tap shift-register-sequence random-number generators. Computers in Physics, **12**(4), 385–392 (1998)